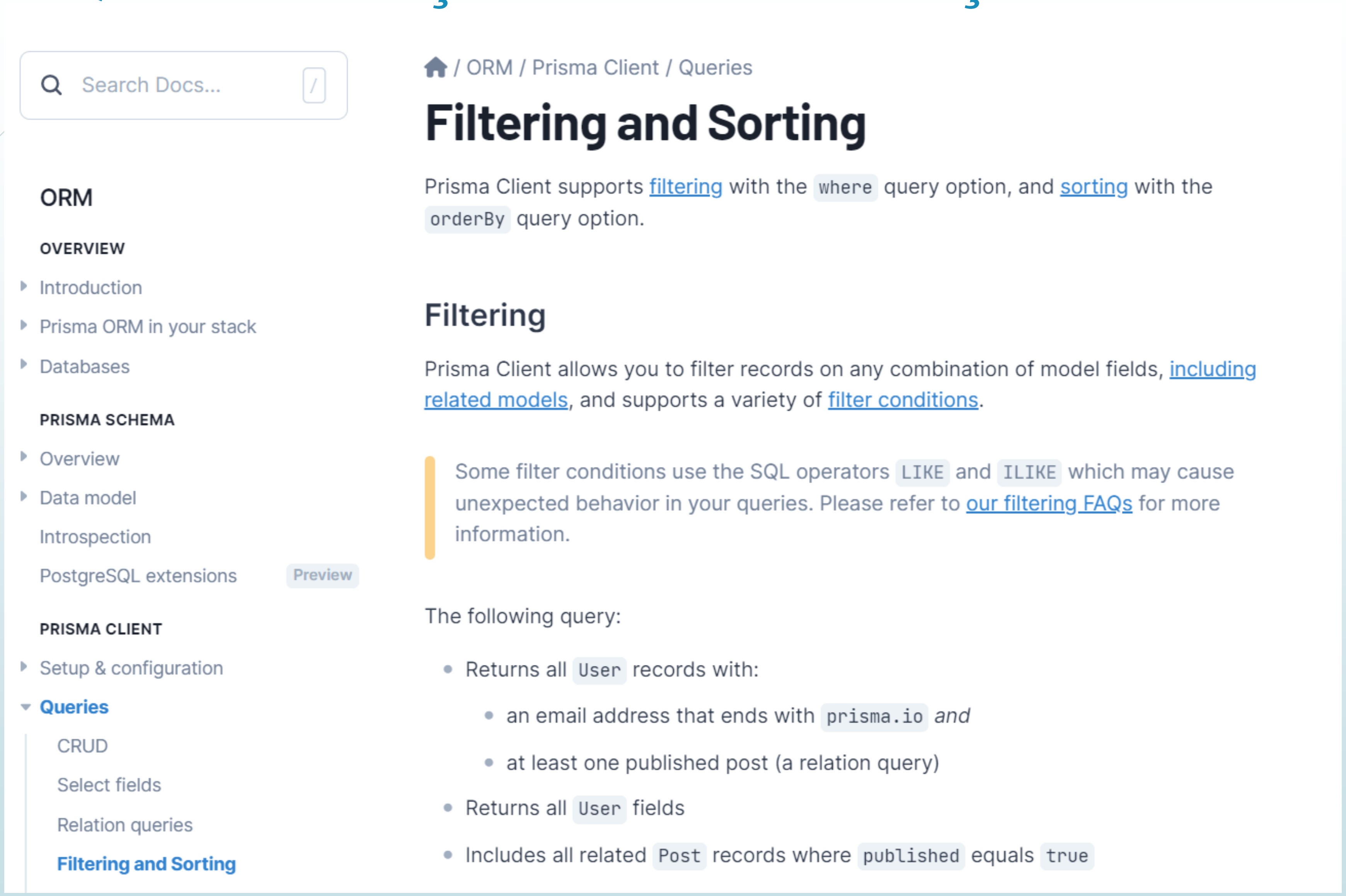
Desenvolvimento de Serviços e APIs

Centro Universitário UniSenac – Campus Pelotas

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Prof. Edécio Fernando Iepsen

Filtros, Ordenações e Totalizações



/ ORM / Prisma Client / Queries

Filtering and Sorting

Prisma Client supports filtering with the `where` query option, and sorting with the `orderBy` query option.

Filtering

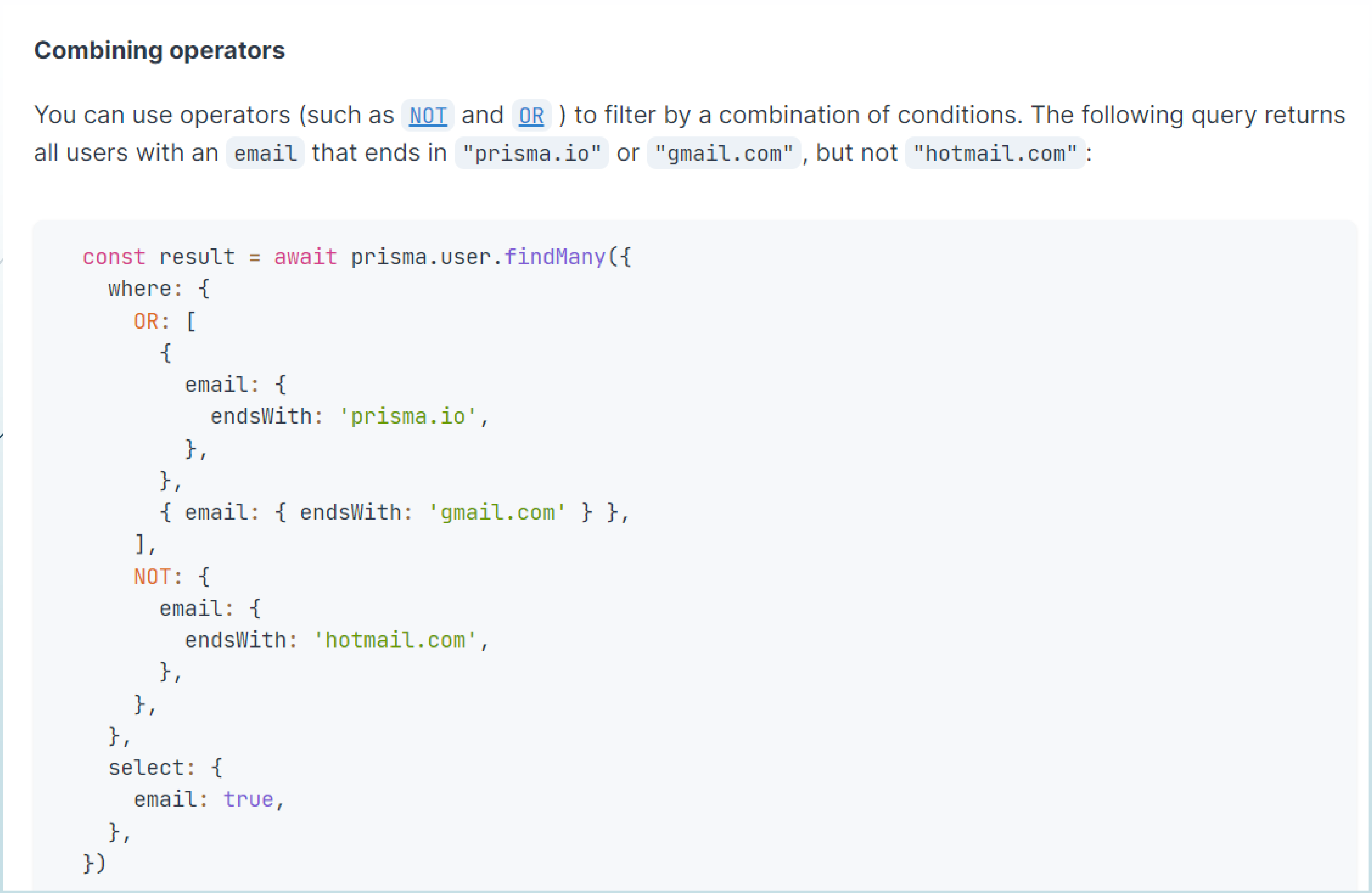
Prisma Client allows you to filter records on any combination of model fields, including related models, and supports a variety of filter conditions.

Some filter conditions use the SQL operators `LIKE` and `ILIKE` which may cause unexpected behavior in your queries. Please refer to our filtering FAQs for more information.

The following query:

- Returns all `User` records with:
  - an email address that ends with `prisma.io` *and*
  - at least one published post (a relation query)
- Returns all `User` fields
- Includes all related `Post` records where `published` equals `true`

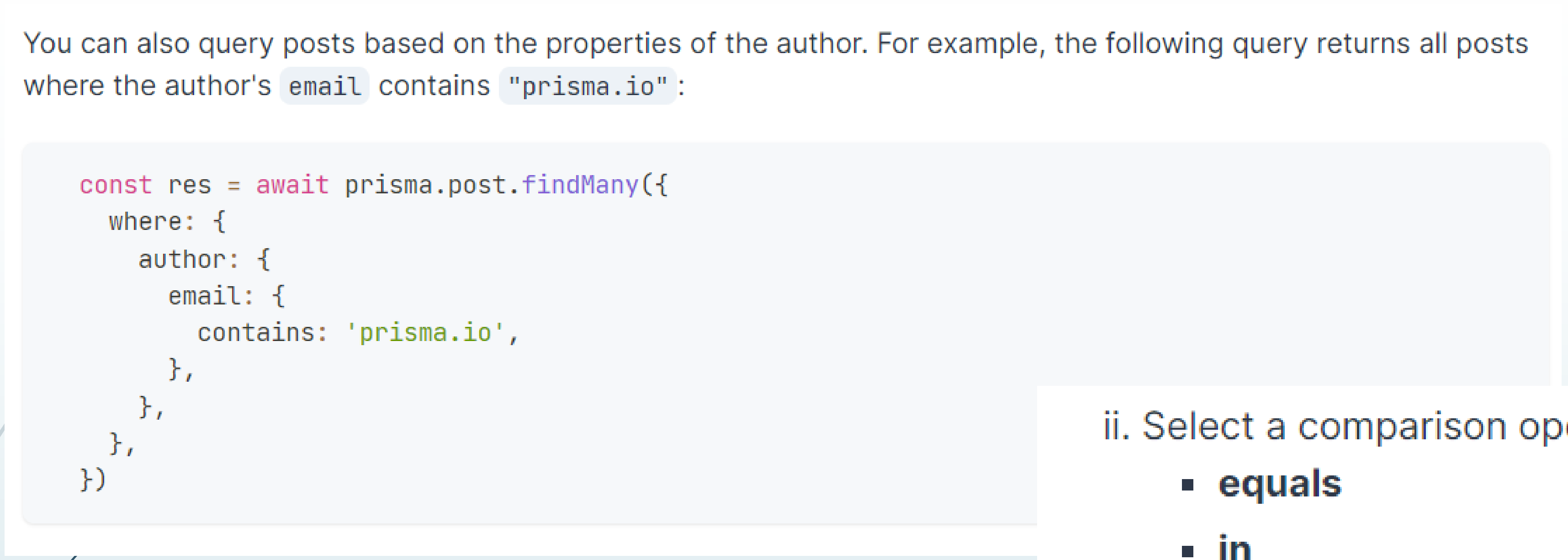
https://www.prisma.io/docs/orm/prisma-client/queries/filtering-and-sorting



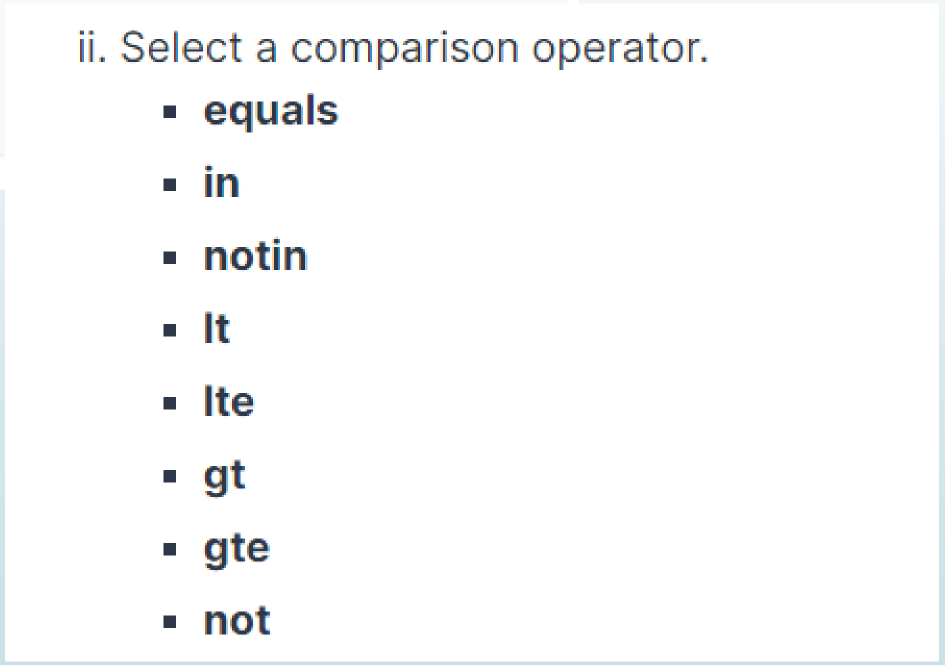
Combining operators

You can use operators (such as `NOT` and `OR`) to filter by a combination of conditions. The following query returns all users with an `email` that ends in `"prisma.io"` or `"gmail.com"`, but not `"hotmail.com"`:

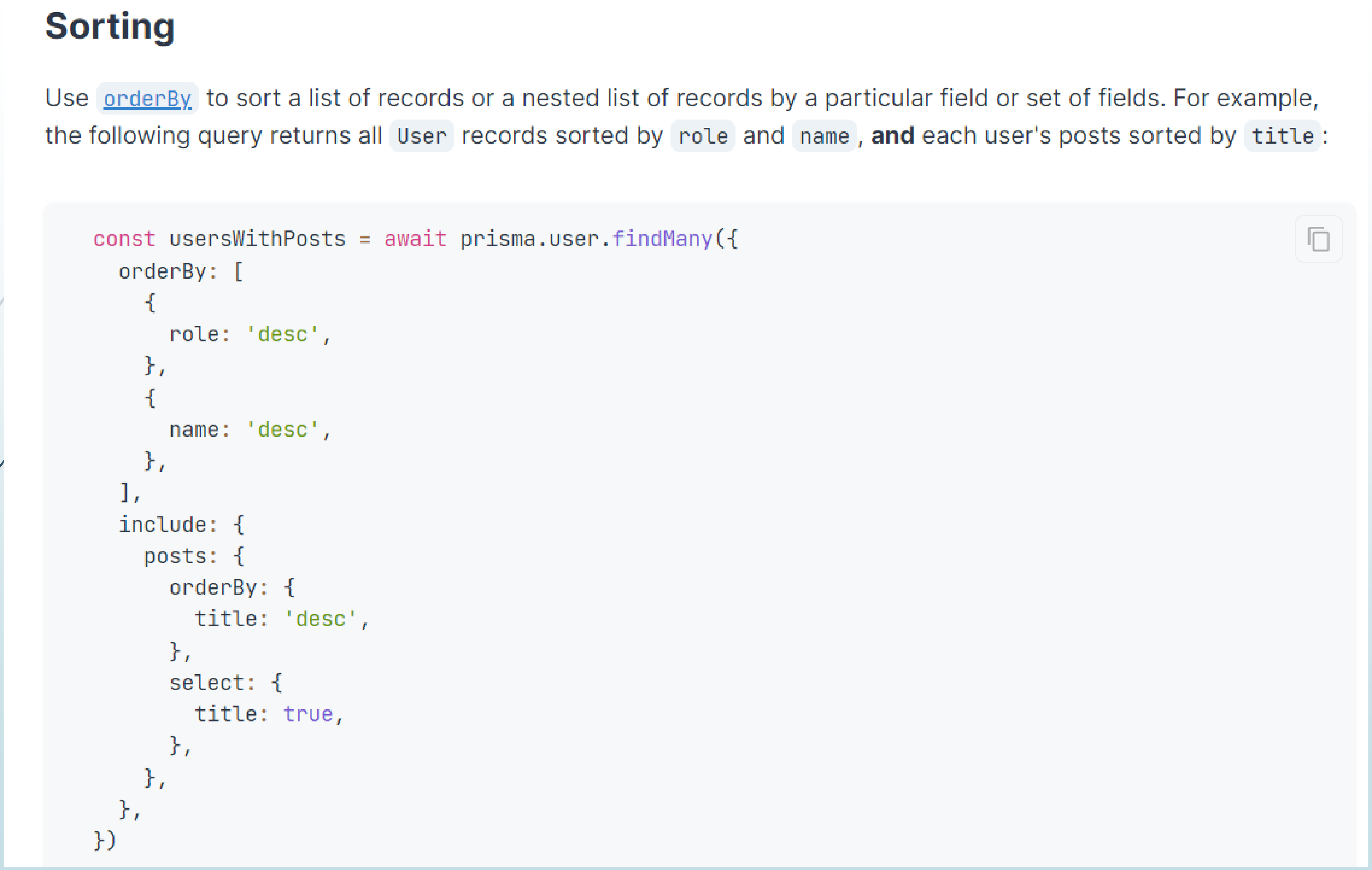
```
const result = await prisma.user.findMany({
  where: {
    OR: [
      {
        email: {
          endsWith: 'prisma.io',
        },
      },
      { email: { endsWith: 'gmail.com' } },
    ],
    NOT: {
      email: {
        endsWith: 'hotmail.com',
      },
    },
  },
  select: {
    email: true,
  },
})
```


You can also query posts based on the properties of the author. For example, the following query returns all posts where the author's `email` contains `"prisma.io"`:

```
const res = await prisma.post.findMany({
  where: {
    author: {
      email: {
        contains: 'prisma.io',
      },
    },
  },
})
```


ii. Select a comparison operator.

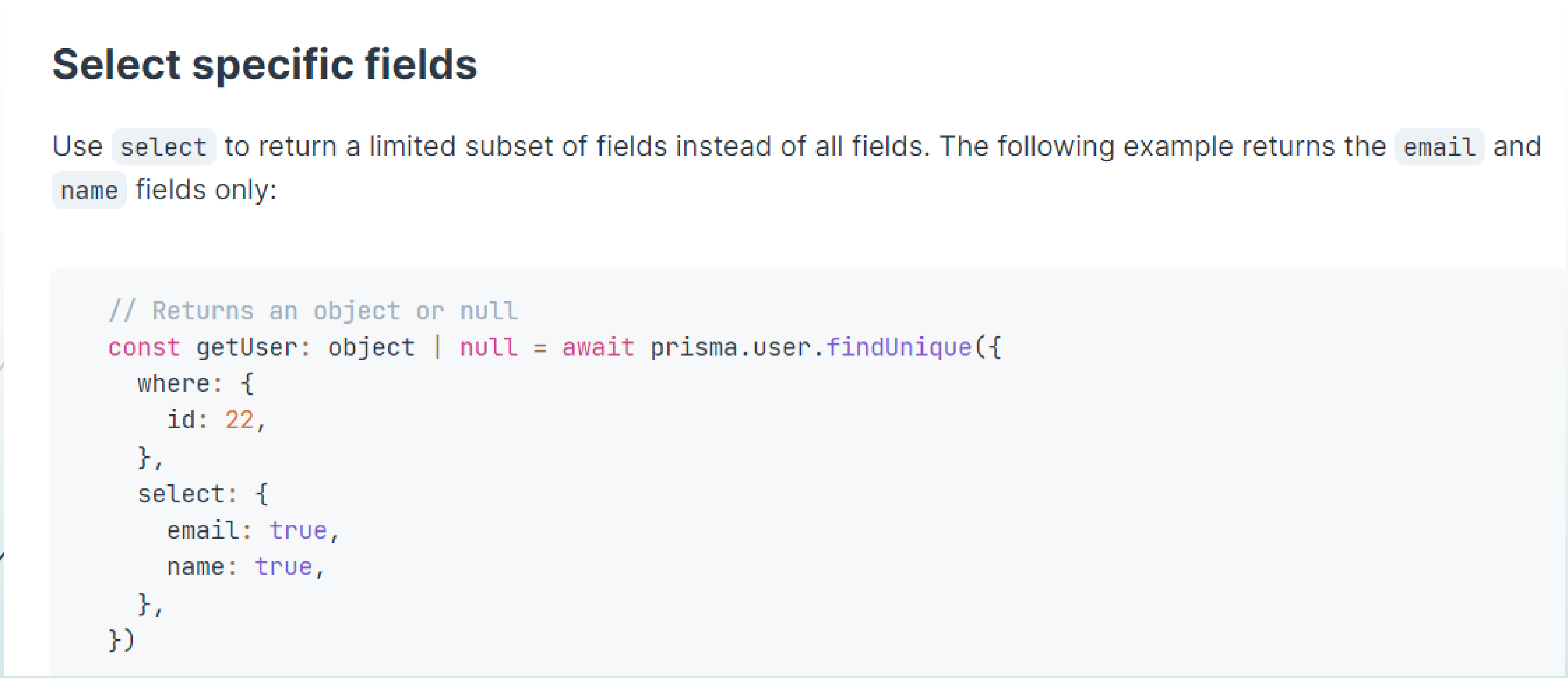
- **equals**
- **in**
- **notin**
- **lt**
- **lte**
- **gt**
- **gte**
- **not**



Sorting

Use `orderBy` to sort a list of records or a nested list of records by a particular field or set of fields. For example, the following query returns all `User` records sorted by `role` and `name`, **and** each user's posts sorted by `title`:

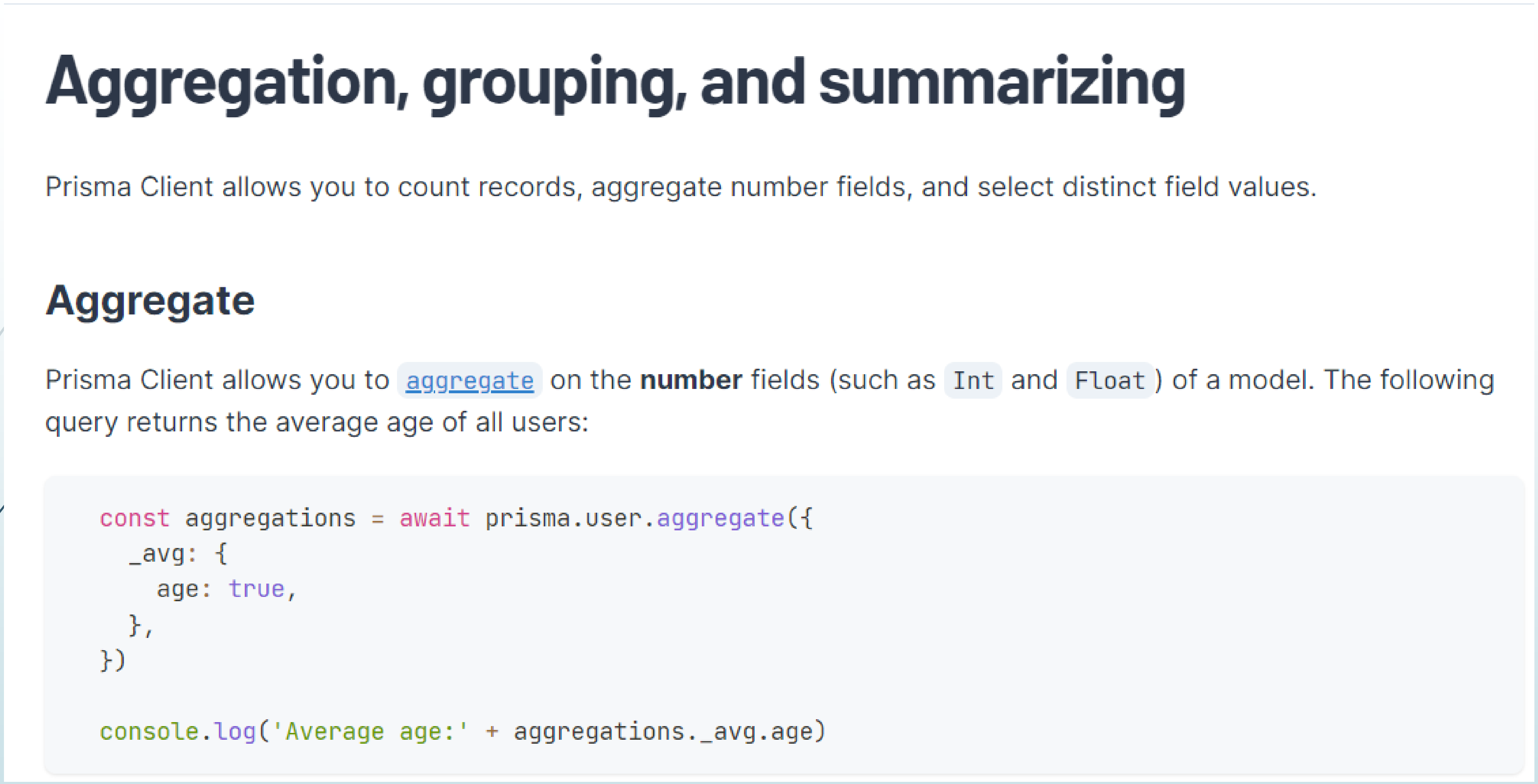
```
const usersWithPosts = await prisma.user.findMany({
  orderBy: [
    {
      role: 'desc',
    },
    {
      name: 'desc',
    },
  ],
  include: {
    posts: {
      orderBy: {
        title: 'desc',
      },
      select: {
        title: true,
      },
    },
  },
})
```



Select specific fields

Use `select` to return a limited subset of fields instead of all fields. The following example returns the `email` and `name` fields only:

```
// Returns an object or null
const getUser: object | null = await prisma.user.findUnique({
  where: {
    id: 22,
  },
  select: {
    email: true,
    name: true,
  },
})
```



Aggregation, grouping, and summarizing

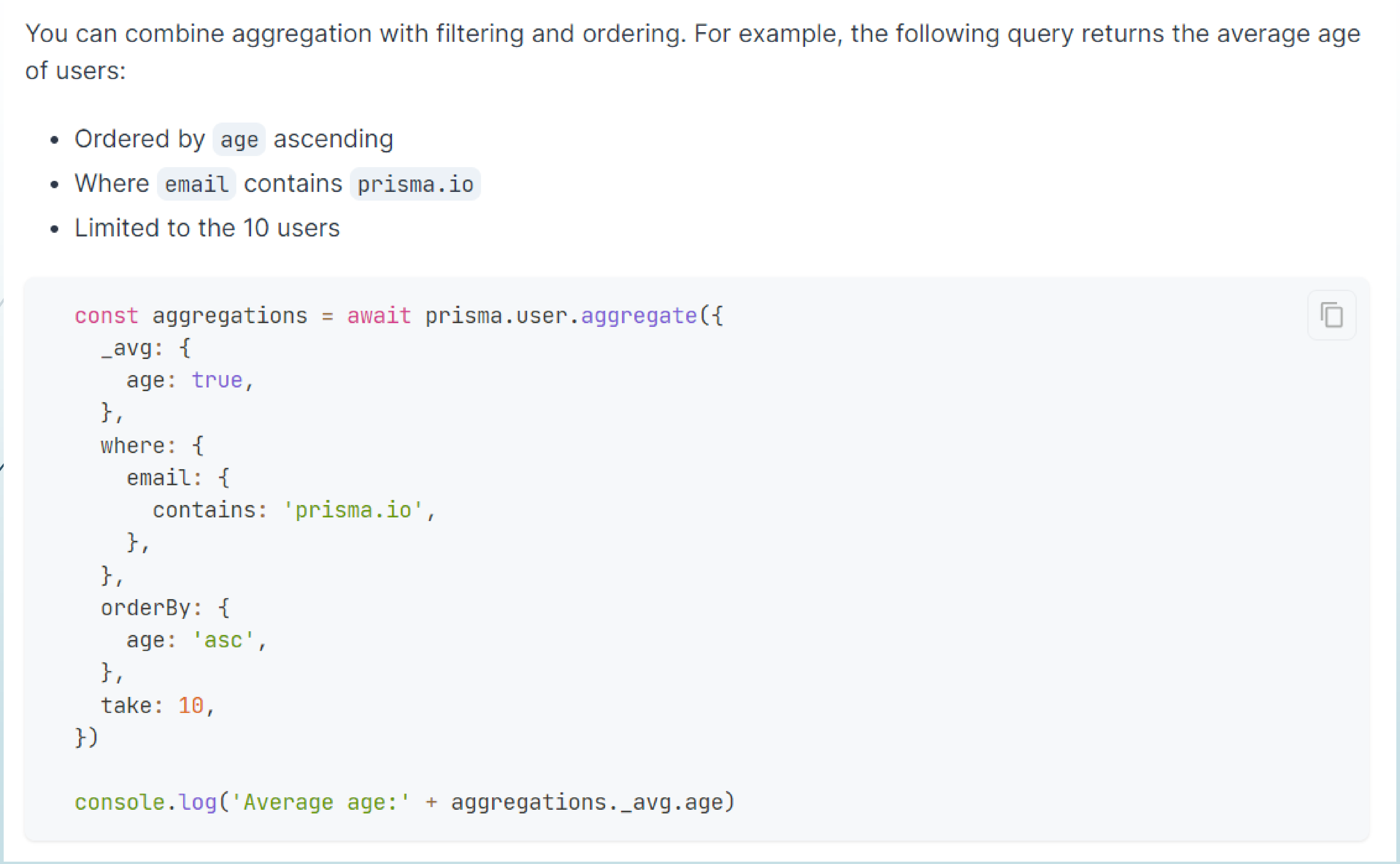
Prisma Client allows you to count records, aggregate number fields, and select distinct field values.

Aggregate

Prisma Client allows you to `aggregate` on the **number** fields (such as `Int` and `Float`) of a model. The following query returns the average age of all users:

```
const aggregations = await prisma.user.aggregate({
  _avg: {
    age: true,
  },
})

console.log('Average age:' + aggregations._avg.age)
```

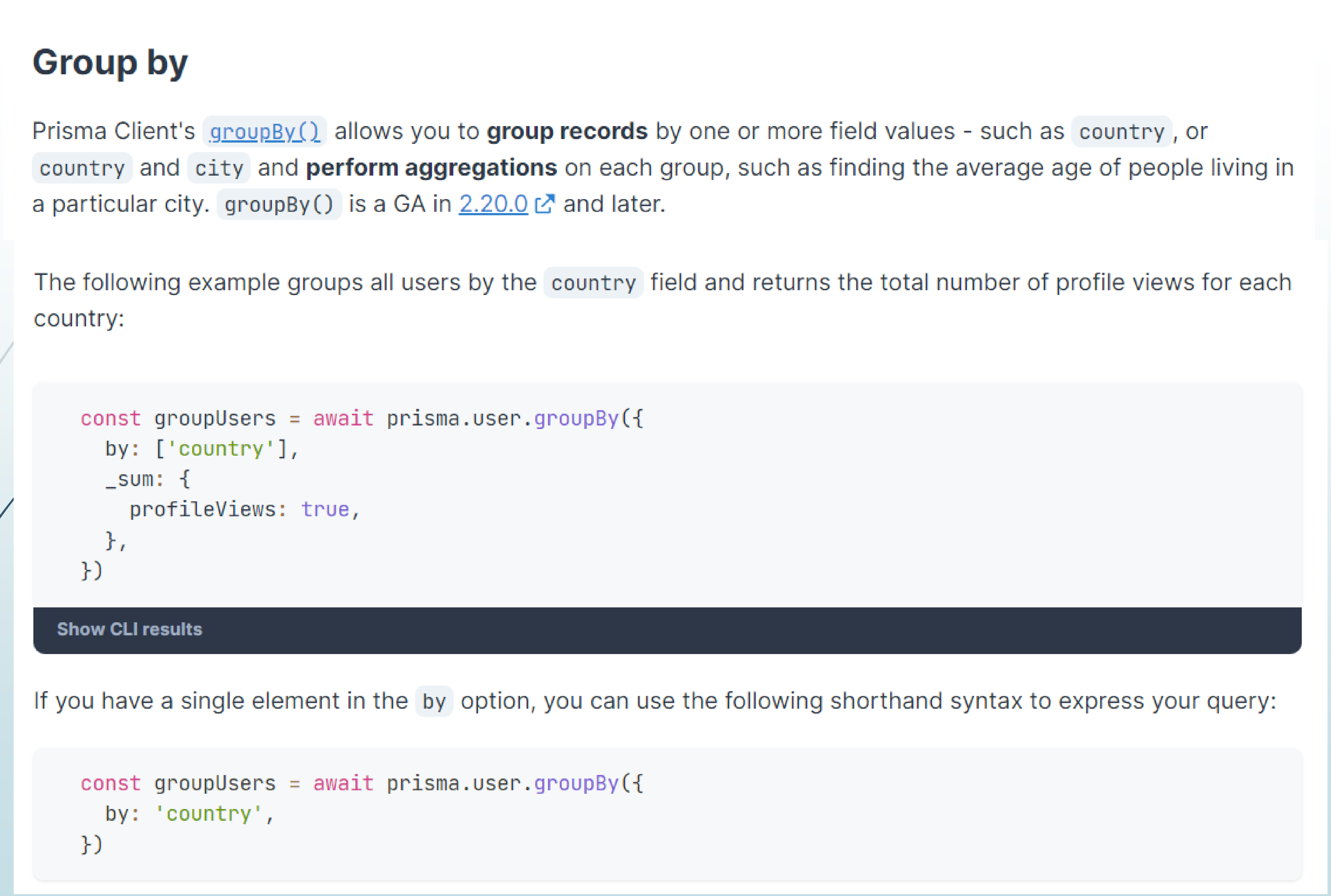


You can combine aggregation with filtering and ordering. For example, the following query returns the average age of users:

- Ordered by `age` ascending
- Where `email` contains `prisma.io`
- Limited to the 10 users

```
const aggregations = await prisma.user.aggregate({
  _avg: {
    age: true,
  },
  where: {
    email: {
      contains: 'prisma.io',
    },
  },
  orderBy: {
    age: 'asc',
  },
  take: 10,
})

console.log('Average age:' + aggregations._avg.age)
```



Group by

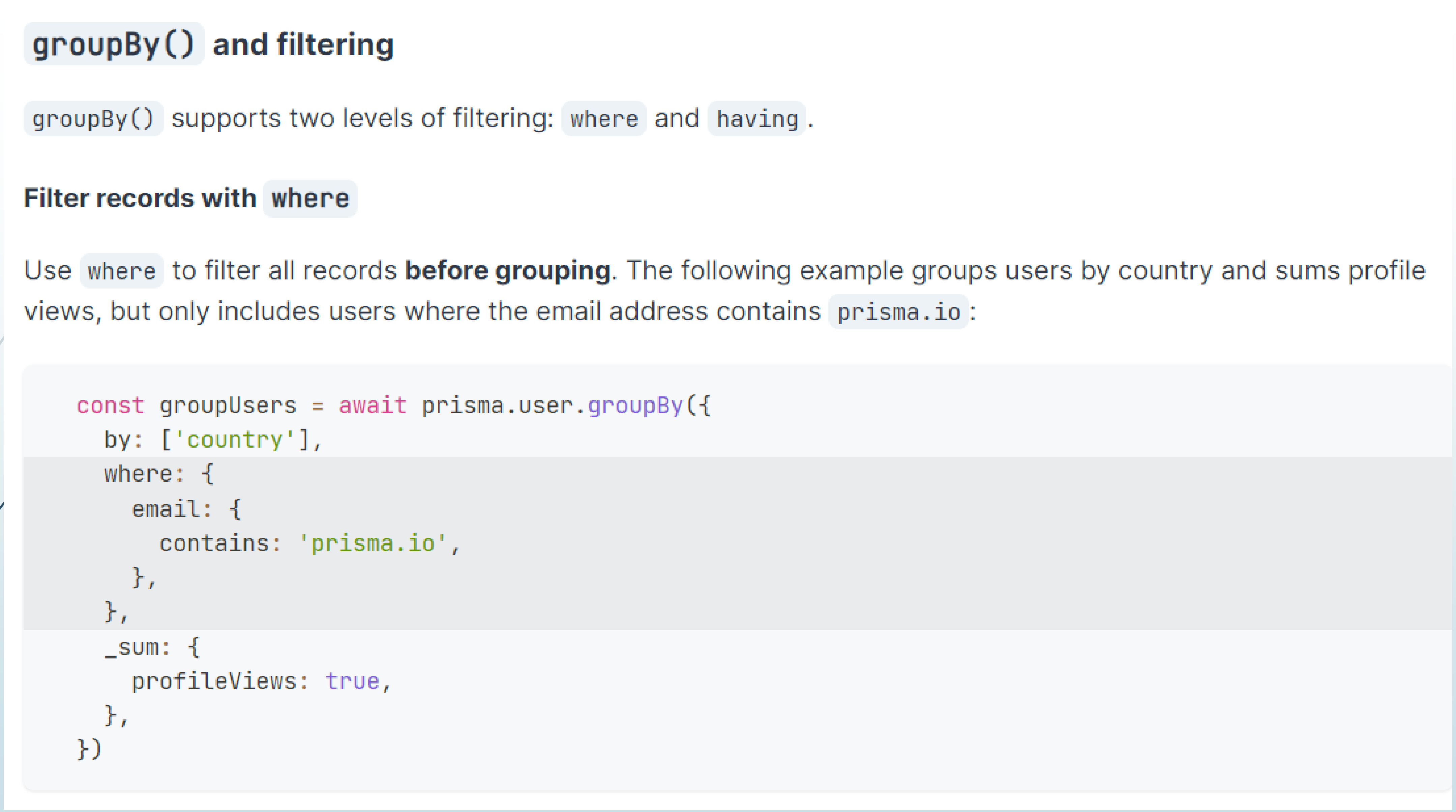
Prisma Client's `groupBy()` allows you to **group records** by one or more field values - such as `country`, or `country` and `city` and **perform aggregations** on each group, such as finding the average age of people living in a particular city. `groupBy()` is a GA in 2.20.0 and later.

The following example groups all users by the `country` field and returns the total number of profile views for each country:

```
const groupUsers = await prisma.user.groupBy({
  by: ['country'],
  _sum: {
    profileViews: true,
  },
})
```

If you have a single element in the `by` option, you can use the following shorthand syntax to express your query:

```
const groupUsers = await prisma.user.groupBy({
  by: 'country',
})
```



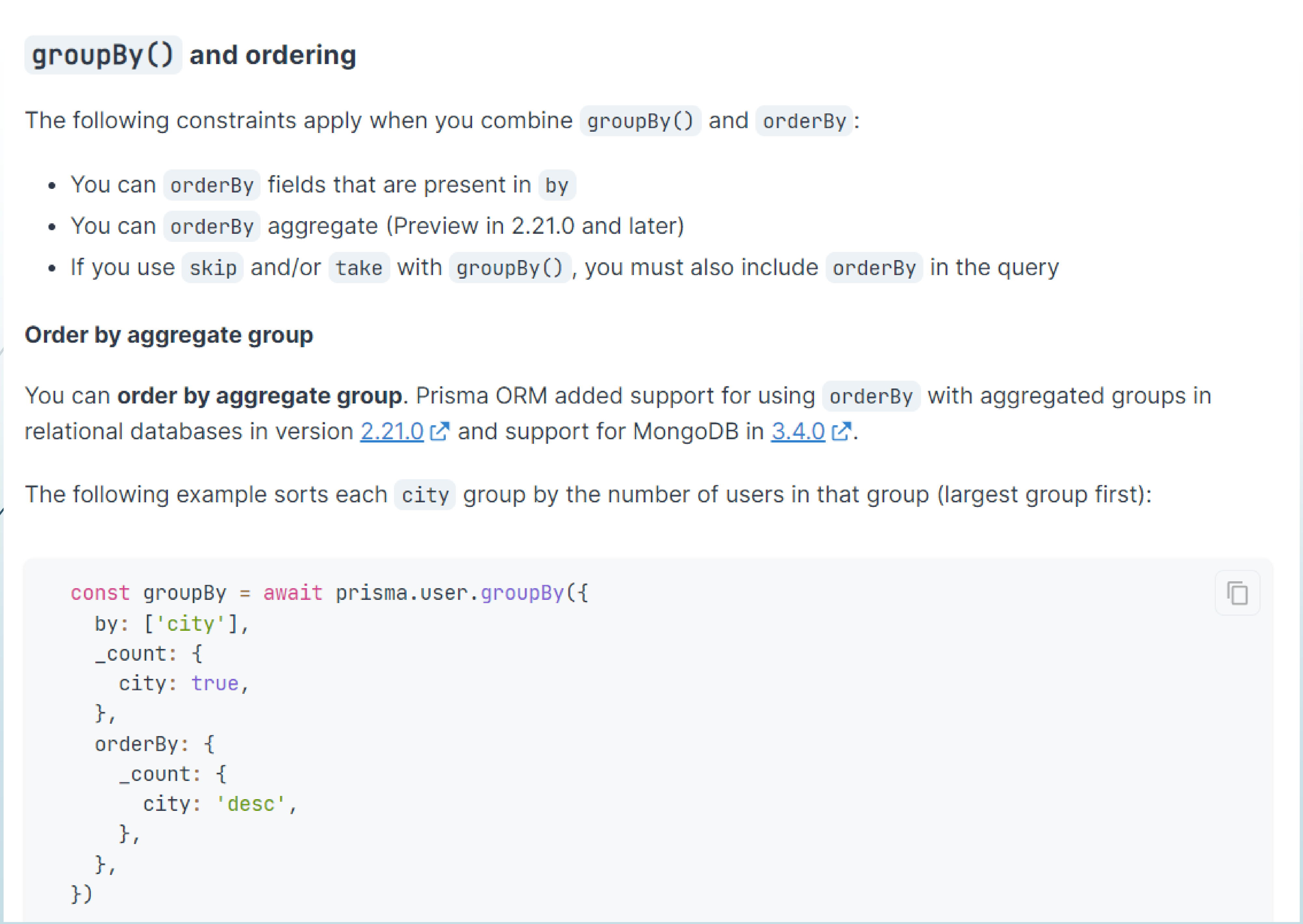
`groupBy()` and filtering

`groupBy()` supports two levels of filtering: `where` and `having`.

Filter records with `where`

Use `where` to filter all records **before grouping**. The following example groups users by country and sums profile views, but only includes users where the email address contains `prisma.io`:

```
const groupUsers = await prisma.user.groupBy({
  by: ['country'],
  where: {
    email: {
      contains: 'prisma.io',
    },
  },
  _sum: {
    profileViews: true,
  },
})
```



`groupBy()` and ordering

The following constraints apply when you combine `groupBy()` and `orderBy`:

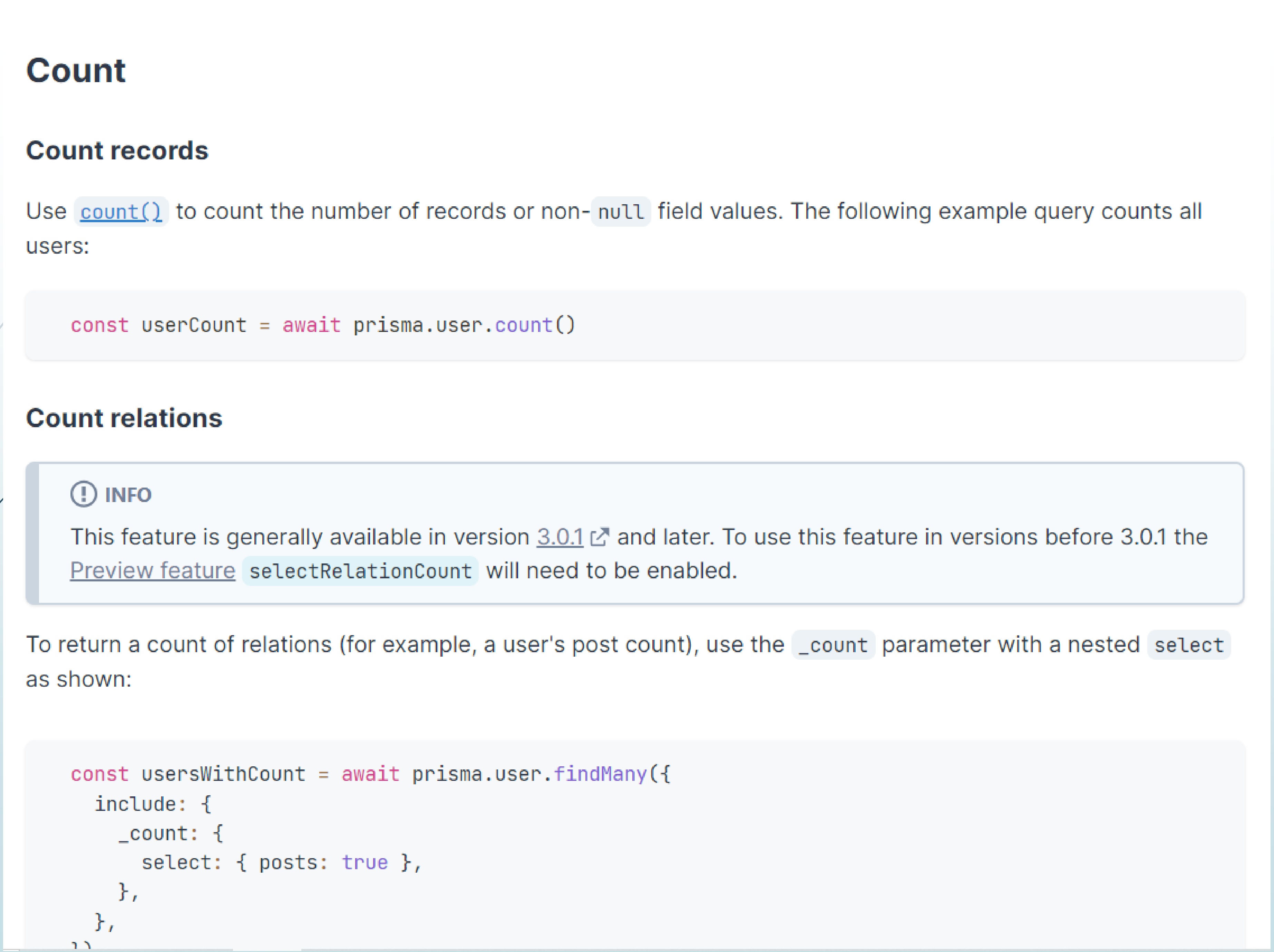
- You can `orderBy` fields that are present in `by`
- You can `orderBy` aggregate (Preview in 2.21.0 and later)
- If you use `skip` and/or `take` with `groupBy()`, you must also include `orderBy` in the query

Order by aggregate group

You can **order by aggregate group**. Prisma ORM added support for using `orderBy` with aggregated groups in relational databases in version 2.21.0 and support for MongoDB in 3.4.0.

The following example sorts each `city` group by the number of users in that group (largest group first):

```
const groupBy = await prisma.user.groupBy({
  by: ['city'],
  _count: {
    city: true,
  },
  orderBy: {
    _count: {
      city: 'desc',
    },
  },
})
```



Count

Count records

Use `count()` to count the number of records or non-`null` field values. The following example query counts all users:

```
const userCount = await prisma.user.count()
```

Count relations

INFO

This feature is generally available in version 3.0.1 and later. To use this feature in versions before 3.0.1 the Preview feature `selectRelationCount` will need to be enabled.

To return a count of relations (for example, a user's post count), use the `_count` parameter with a nested `select` as shown:

```
const usersWithCount = await prisma.user.findMany({
  include: {
    _count: {
      select: { posts: true },
    },
  },
})
```